CHAPTER FOUR

Mechanics of Game Feel

To wrap up our section on defining game feel, let's apply all the ideas from Chapters 1, 2 and 3 to some specific games. To do this, we'll return to our three-part definition of game feel: real-time control, simulated space and polish. The overall question to be answered is where a game fits on the diagram (Figure 4.1).

This breaks down, once again, into three questions:

1. Does it have real-time control?
2. Does it have simulated space?
3. Does it have polish?

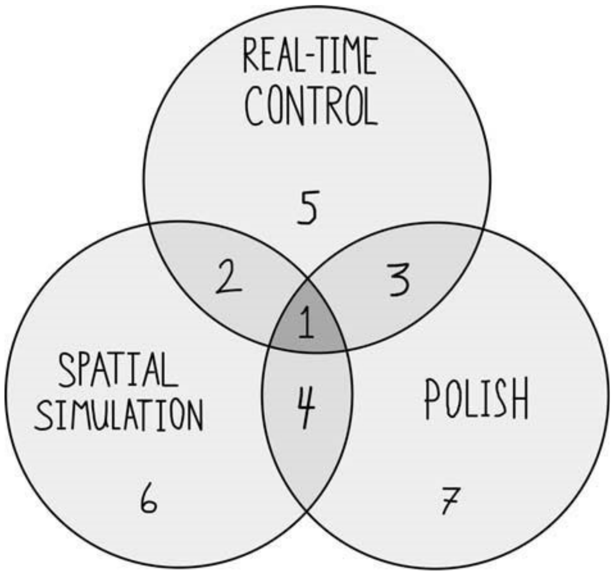

FIGURE **4.1 Types of game feel: we want to put every game somewhere on the diagram.**

From our model, we have measurable thresholds for real-time control to test against:

- 10 frames per second. The images are displayed at a rate faster than one cycle of the human perceptual processor, which will be 50 to 200 ms. Therefore, images displayed at a rate at or above a rate of 10 frames per second will appear fused into motion, and 20 frames per second or higher is necessary for a smooth motion. In the case of a game, this is not a series of linear frames played back in sequence but a series of states generated in response to input.
- Response time of 100 ms or less. The game's response to input happens within one perceptual cycle (50 to 200 ms) of the player's action, fusing into a sense of causality and instantaneous response.
- A continuous feedback loop. The game provides a continuous, unbroken flow of input and instant response, enabling ongoing correction cycles to occur.

But these metrics are difficult to apply to an entire game's interactivity all at once. To answer the question of real-time control more easily, it's useful to break down a game's interactions into individual mechanics. Then we can check each mechanic against the various thresholds from our model.

Mechanics: Game Feel Atoms

For our purposes, a "game mechanic" is one complete loop of interaction, such as a single mouse twitch, button press or foot stomp that can be traced through the game's programmed response and back to the player over and over again. Another way to think about mechanics is as verbs. What are the player's abilities in the game? What can the player do? By this definition, examples of individual mechanics include:

- Pressing the A-button to jump in Super Mario Brothers
- Steering Mario left and right using the D-Pad in Super Mario Brothers
- Strumming a note in Guitar Hero
- Using the mouse to steer left and right in flow
- Boosting forward by clicking the mouse in flow
- Drag-selecting a group of units in Starcraft
- Clicking to send a group of selected units to a new location
- Pressing a button at the right moment to advance to the next sequence in Dragon's Lair
- Clicking on a button to select the next technology to research in Civilization 4

In a typical game, many different mechanics are active at the same time and often overlap and combine. Running and jumping in Super Mario Brothers are separate mechanics by our definition. But they combine to become a whole greater than the sum of their parts.

Mechanics can also change over the course of a game. For example, gaining a skill point in Tony Hawk's Underground makes your skater move forward faster. Mechanics can also come and go over the course of a game. In Half-Life, the player finds new weapons gradually, but then loses them at the halfway point of the game. Super Metroid gives the player a bunch of mechanics up front, then takes them away a short time later, forcing the player to start from scratch.

The question is whether or not each individual mechanic meets the criteria of real-time control and whether the system as a whole sustains real-time control.

Knowing whether a game has real-time control is most of the challenge. From there, it's just a matter of asking whether the game has literal simulated space, if the player perceives that space actively, and if the polish effects are used to emphasize physical interactions in that space.

Applying the Criteria

To put our definition to the final test, we'll apply it to four games: Street Fighter II, Prince of Persia, Guitar Hero and Kirby: Canvas Curse. Each game is on the fringes of our definition in its own way.

Street Fighter II

There are three primary types of mechanics in Street Fighter II: walking, attacking and jumping. The walking mechanic responds within 100 ms when the joystick is moved left or right, and it allows a sustained correction cycle. Input is constantly accepted, the game responds within 100 ms, and there is no lockout period. As soon as I perceive the result of my last action, I can adjust it with a new input. The movement mechanic has real-time control.

The "attack" mechanics—when the player presses one of the six attack buttons—have interrupted continuity. Pressing a button plays back an animation, which changes the shape of the avatar. The response time when the button is pressed is instantaneous, but then the player is locked out of further input until the animation is complete. For the "light" attacks, the duration is very short and will not interrupt the correction cycle of the walking mechanic. The heavy attacks can take almost one second to complete, however, disrupting the continuity of control. Either way, pressing a button to trigger an animation is not a continuous correction cycle. The attack mechanics do not have real-time control.

The jump mechanic adds upward force to the player when the joystick is pressed up. Once the jump has started, the player cannot alter the trajectory of the jump. This temporarily takes control away from the player, breaking the correction cycle of the movement mechanic. After leaving the ground, however, the player can still trigger attacks. This mitigates the fact that the player's correction cycle is temporarily

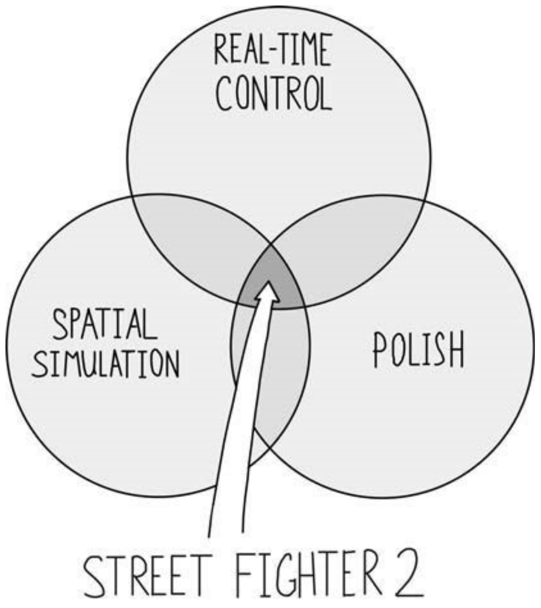

FIGURE **4.2 Street Fighter II has game feel.**

broken, as does the fact that the player gets to choose when to start the jump. The player feels they have real-time control over everything. The whole system, combining the movement, jumping and attack mechanics, has real-time control.

Street Fighter II also has simulated space. The characters collide with the ground, the edge of the screen and with each other. These interactions are perceived actively by the player, through the correction cycle of the movement mechanic.

Finally, the polish effects in Street Fighter II—the sounds, particle effects and animations—emphasize the interactions between objects in the game world.

Prince of Persia

The original Prince of Persia (see Figure 4.3) is an interesting edge case because of the disconnect between animation and control. The character moves fluidly, but the feel of control is stilted and uneven. A casual observer might assume that because the movement of the character is smooth and even, that the control must also be. This is not the case.

The individual mechanics in Prince of Persia are:

- Run
- Jump vertically
- Jump horizontally

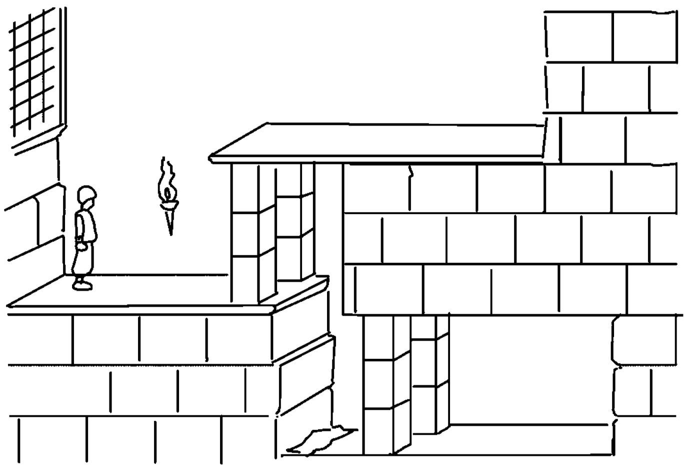

FIGURE **4.3 In Prince of Persia, animation reigns supreme.**

- Change direction
- Lower (down a ledge)
- Draw sword
- Sheathe sword
- Shuffle
- Parry
- Thrust
- Crouch
- Crouch-hop
- Walk
- Grab ledge
- Crouch-slide

Prince of Persia consists entirely of mechanics like the attack mechanic in Street Fighter II. The player presses a single button, and a single animation is played back. The response time is less than 100 ms, but further input is locked out until the animation is over, which often breaks continuity. Examining individual movement mechanics by this criteria, we can see which ones have real-time control and which don't. For example, going from Stand to Run (Figure 4.4) fails one of our threshold tests:

It takes almost 900 ms for the prince to go from standing still to a full speed run. In between, new input from the player is meaningless. There is a branch point of sorts; having reached the end of the "standing to run" animation, if the directional

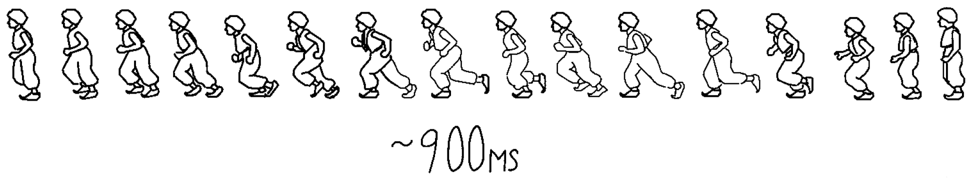

FIGURE **4.4 Sixteen frames at 30 fps = 0.53 seconds to complete the animation.**

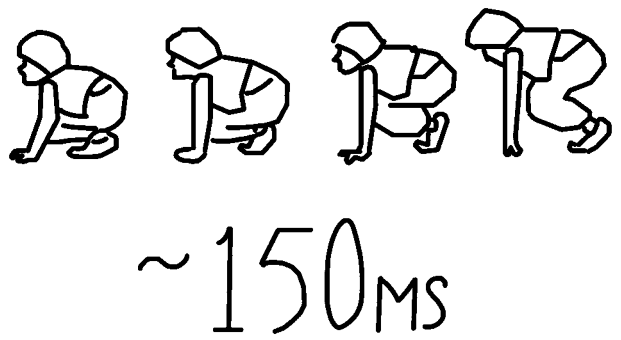

FIGURE **4.5 The hopping animation takes only 150 ms to complete, so it feels almost real time.**

button is still held down, the prince goes into his full speed run cycle. If the button is not held down, the "run to standing" animation is played back, taking another few hundred milliseconds. This is not an unbroken correction cycle, so this particular mechanic by itself does not have real-time control.

Only the crouching mechanic, comprised of the fewest number frames, has real-time control. Because the user is locked out for a very short amount of time, the action not only feels instantaneous in response, but it feels as though it's ready to accept new input as soon as the player is ready to offer it. It's no wonder, then, that this is the mechanic of choice to use when precision timing is necessary. When navigating through a room full of gnashing blades, you want to use the crouch-hop mechanic (Figure 4.5). It feels like the most precise and responsive expression of your input and enables the smallest increments of movement spatially.

Out of all the mechanics of Prince of Persia, only one passes our threshold tests for real-time control. But the animation is fluid and appealing and covers up the lack of control to some degree. The player rarely has a sustained correction cycle and so rarely experiences true game feel. The fact that there are interactive branch points in the animations helps to some degree. In this case, unpredictability actually works in the game's favor. I don't know exactly when the jump is going to take place, so I instinctively just hold the up button when I'm close to where I want to jump. This makes me feel as though the system is listening to my input more often than it is.

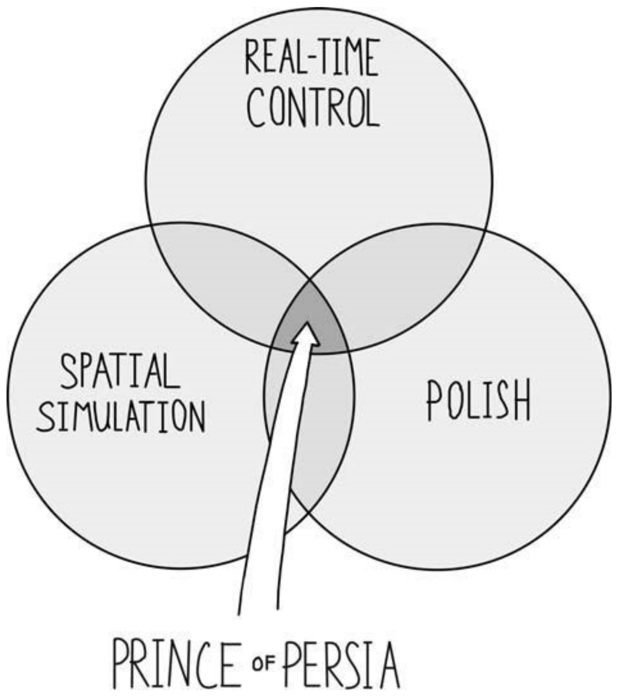

FIGURE **4.6 Prince of Persia has game feel, but it's pretty stilted.**

There is simulated space in Prince of Persia; animations can be interrupted when the character walks far enough off a ledge, and he can bump into walls. The player experiences these directly and actively, by pushing the character into them. The only polish effects that emphasize these interactions are the animations, which have a good sense of weight and presence against the floor.

So Prince of Persia has game feel, but just barely. The player is able to cobble together a correction cycle by imagining control when there is none and by using the mechanics with the lowest number of frames whenever possible.

Guitar Hero

Ahh, Guitar Hero. What a lovely game. It's rare to see technology infused with such a sneer, such sense of unabashed glee. In the Game Developer post mortem of the game, producers Greg LoPiccolo and Daniel Sussman name the one litmus test for every feature and piece of content in the game: "Does it rock?" The results of this simple vision speak for themselves. But does Guitar Hero have game feel as we've defined it? Again, let's examine the individual mechanics and the system as a whole.

In Guitar Hero, there are five things you can do (mechanics):

- Strum
- Whammy
- Hammer On
- Hammer Off
- Tilt

The strum is the game's core mechanic (Figure 4.7). Colored notes scroll down from the top of the screen. You hold down one or more corresponding buttons on the neck of the plastic guitar and pull the strum trigger up or down. If you strum the right combination of notes at the right time (when the note's position is close enough to crossing the line) the game records the note as hit. More notes hit means a better score, and the game tracks streaks of hit notes. Miss too many notes and you fail the song.

Impression of motion, check. The notes seem to move down the screen, from top to bottom, and individual frames are fused into an impression of moving objects. Instantaneous response, check. The response time to input is within one perceptual processor cycle (less than 100 ms) so the response from the system seems instantaneous with a strum. But there is no continuity. Instead of locking the player out as in Prince of Persia, it cuts the player off. The whole loop of input and response happens in less than 100 ms, but once it's done it's done. There is no continuous flow of input and response, no correction cycle.

The whammy bar mechanic, however, allows a constant stream of both input and response. The response feels instantaneous and continuity is maintained. The whammy mechanic has the potential to be an ongoing correction cycle. But there

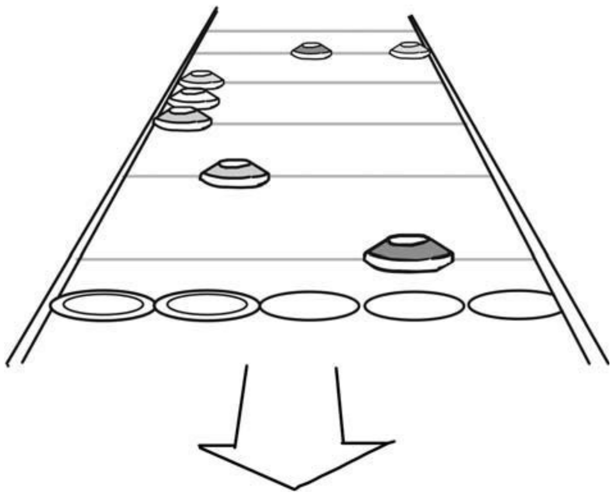

FIGURE **4.7 The "strum" mechanic in Guitar Hero.**

is no simulated space. The waveform ripples and notes bend as the whammy bar is manipulated, but the size of the ripples has no meaning. Bending notes with the whammy mechanic does not enable the player to actively perceive a simulated space because there is no simulated space around it to interact with (Figure 4.8).

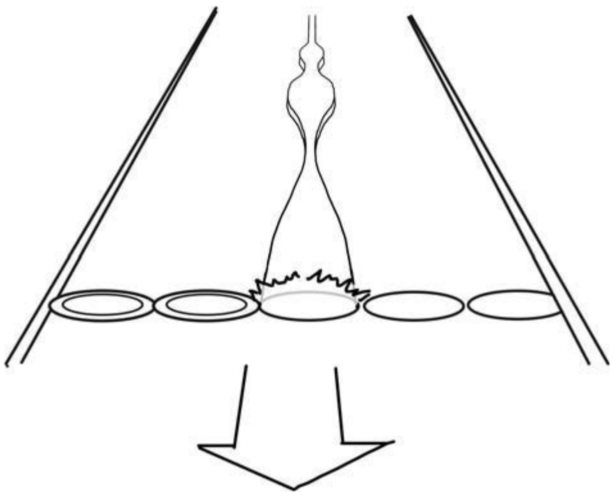

FIGURE **4.8 Bending the waveforms with the whammy mechanic is real-time control, but it lacks spatial simulation.**

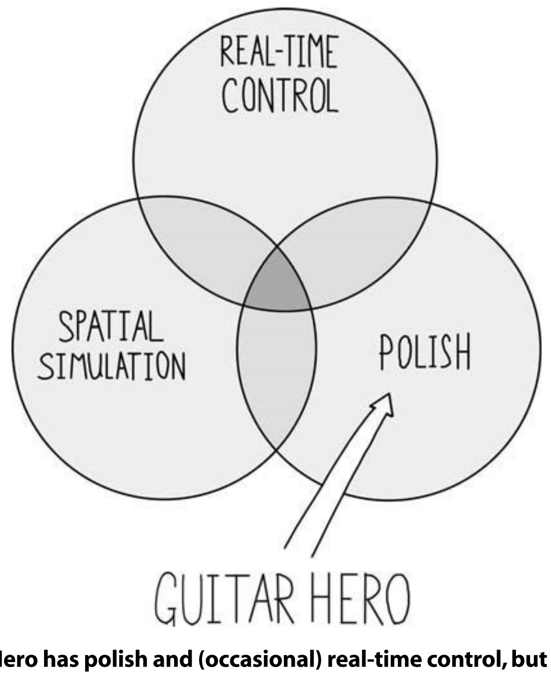

FIGURE **4.9 Guitar Hero has polish and (occasional) real-time control, but no simulated space.**

Guitar Hero is a relatively simple game. Strumming to hit notes in increasingly difficult patterns that are synched to songs is the vast majority of the game. Even considered as a whole system, however, it does not have the property we've defined as game feel. The notes may fly fast and furious, and you can wail on the whammy and tilt to use your star power, but there is no unbroken flow of action, perception and contemplation. There is the impression of motion, instantaneous response, but there no sustained correction cycle and no spatial simulation (Figure 4.9).

Kirby: Canvas Curse

Kirby: Canvas Curse (Figure 4.10) takes a very simple idea and executes on it brilliantly, enabling the player to indirectly control Kirby's movement by drawing. In Canvas Curse, you play as Kirby and as a disembodied paintbrush at the same time. There are three mechanics:

- Drawing (paintbrush)
- Tapping (on the avatar)
- Holding (enemies)

Using the paintbrush mechanic, you draw lines on the screen, represented by flowing rainbows. If Kirby comes into contact with these lines he will follow their path in the direction they were drawn (see Figure 4.11).

From the moment the player starts drawing the line, they're running a correction cycle to get the line drawn in the shape and direction they want. The response is instantaneous, but this is not real-time control per our definition. In this case, the DS stylus and screen are functioning the same way a piece of paper and pencil do.

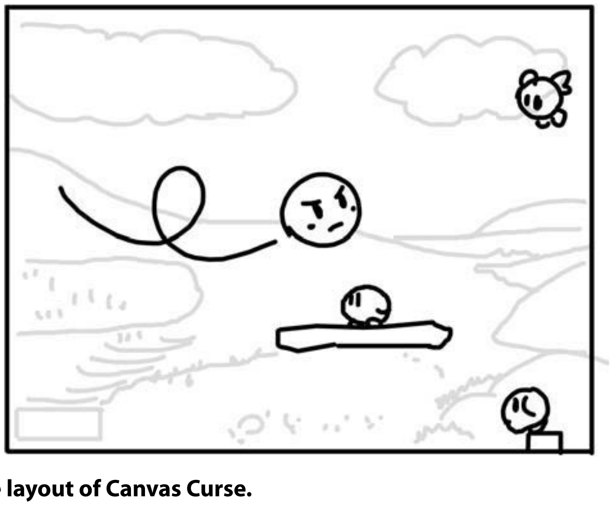

FIGURE **4.10 The layout of Canvas Curse.**

The player is correcting the movement of his or her own hand in space rather than a virtual object in virtual space.

The other main mechanic is tapping. The player can tap Kirby directly with the stylus. This results in a state change and a speed boost. The spinning animation is accompanied by a burst of speed in the direction Kirby is currently facing. The response is instantaneous, but, as with Guitar Hero, the input is not sustained.

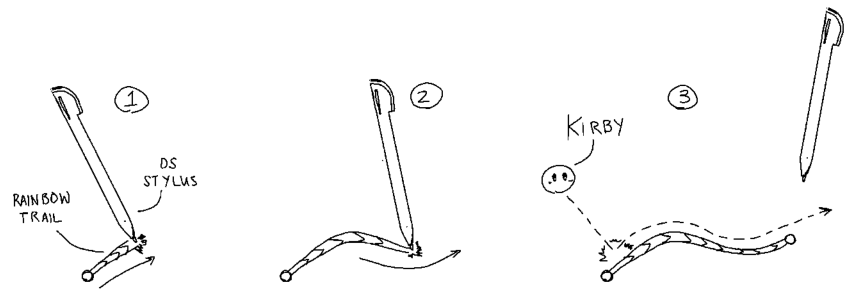

FIGURE **4.11 As you draw the rainbow trail, Kirby will follow its path if he makes contact with any part of the trail.**

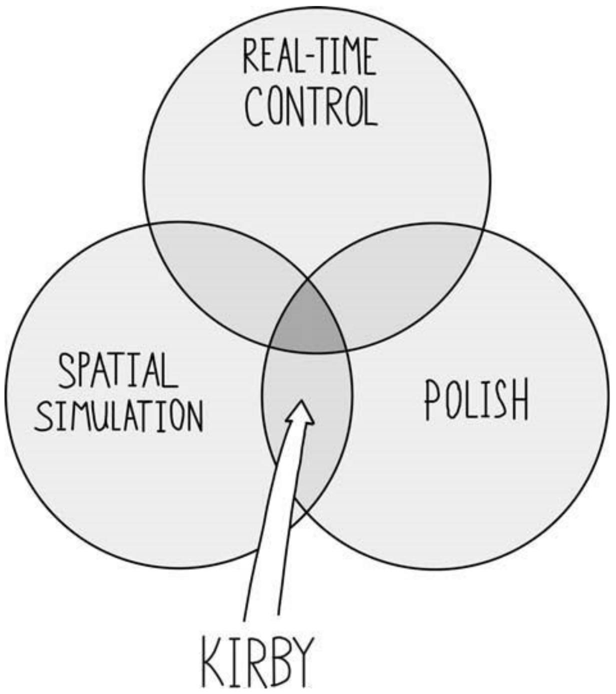

FIGURE **4.12 Canvas Curse has polish and simulated space, but no sustained real-time control.**

One tap equals one input. To express a new input, the player has to lift the stylus off the screen and tap again. This is not an ongoing correction cycle.

The ambiguity of Canvas Curse is in its simulation. Kirby moves around a simulated space, colliding with walls, enemies and other objects. Those interactions are emphasized with polish effects like sounds and particles. This is where things become fuzzy; Kirby interacts with simulated space in just the way it should to fall inside our definition of game feel. The world of Canvas is its own unique physical world. But the player does not experience the simulated space directly, perceiving it actively via Kirby's "body" in virtual space. Instead, the player guides Kirby indirectly, observing the results of his interactions and building a model of the game world from those interactions. Kirby: Canvas Curse falls outside the definition of game feel (Figure 4.12).

Summary

Breaking down game feel into its component mechanics on a game-by-game basis enables us to better understand which games have game feel and which don't, and why. Our definition is now complete: even games that are on the edge can be classified according to real-time control, simulated space and polish. Games that have these three properties have game feel. Games that don't fall outside the scope of this book.

CHAPTER FIVE

Beyond Intuition: Metrics for Game Feel

In this chapter and the next six, we'll explore the problem of measuring game feel. The goal is to compare the feel of one game to another meaningfully. This will give us a generic, working vocabulary for game feel and will offer insight into why some games feel good and others do not, even if the games appear similar on the surface. In Chapter 1, we defined the canvas of game feel and looked at some of the finished "paintings" of experience possible on that canvas. Now our goal is to identify the colors of paint.

Why Measure Game Feel?

As with definition, there are no standard measures for game feel. We as players and designers do not attempt to measure game feel or to compare the feel of one game to another at a level deeper than is necessary for casual conversation and game production. From players, we have vague descriptions like floaty, loose, tight and responsive. Some enlightened game designers measure response lag and move timings, but to most game feel tuning is intuition. When a game designer sits down to create a mechanic from scratch, this is a problem. As God of War designer Derek Daniels says, "One of the worst things about making video games is that you have to re-invent the wheel with almost every new project you work on. So even though Mario jumps like a champ, when you go to make your game, it's very hard to reverse engineer Mario's jump and port it into your game."[1]

This is frustrating because each new game we design feels just as complex as the last. If we don't directly copy what we did before, we're starting from scratch.

[1]http://lowfierce.blogspot.com/2006/05/why-some-games-feel-better-than-others.html

This leads to timid, incremental improvements over previous designs in an effort to keep things safe and comfy. We copy Mario's mechanics, or Banjo Kazooie's, or Grand Theft Auto's, trying to recreate, in the context of our own systems, what was good in those. This is the easy way to design.

The more difficult way is to ask the following question: Where did the feel of Mario come from? Or Spacewar! for that matter? Those designers didn't have something to clone from, so how did they arrive at good-feeling mechanics? We continue to use a handful of games as our exemplars of mechanic design and game feel. Yet we fail to identify what is special about these particular combinations of real-time control, simulated space and polish. We need to understand the unique relationships between the parts and how these relationships give rise to the experiences we cherish.

There are common elements in the physical design of a controller, the relationship between physical and virtual movement, and the design of the virtual worlds we interact with through game feel. If we can identify and measure these pieces of game feel, we can avoid constant reinvention. This requires us to wrap our brains around the game feel system as a whole—including the player, the input device, the programmed reaction from the game system and all the pieces (i.e., the Game Feel Model of Interactivity described in Chapter 3)—and identify which elements enable us to make a meaningful comparison between the feel of two games. Not only the pieces, but the relationships between the pieces. If we can do that, we can understand how to construct similar systems by insight instead of emulation.

Soft Metrics vs. Hard Metrics

Before diving down into specific elements from the Game Feel Model of Interactivity that we're going to apply as metrics, a few words on measuring game experiences. As every designer knows, the only valid way to take the temperature of a design in progress is to watch players play it. There's no way around it; the output of a game system is player experience. To master it, you've got to measure it. To measure it, you need live players.

Enter the dreaded play test. Nothing is more humbling. You have a vision in your head of the experience the player should have playing the game. Perhaps this vision lines up with the experience you as the game designer currently have when playing your game. Perhaps you feel exceptionally skillful and adept as you play and you think that playing your own game is pretty fun. Now put the game in front of some players and watch as your tower of hope gets hit by the oily wrecking ball of player reality. The players do unexpected things, are hung up on stupid little details or can't figure out the controls. They whine, they grunt, they say "this is stupid!" and they walk away in disgust. They ignore all instructions, mash their way around, and display all the insight and cognitive capacity of an indignant end table. They always tell you, "Oh, it's fun, but …" and list a litany of bizarre sounding changes they'd like to see. Brutal!

And it gets worse: this is not the player's natural environment. If you weren't standing there, if you hadn't invited or cajoled, or if the playtester weren't your friend or sibling or spouse, would they still be playing the game? The answer is usually no. The closer you can get to seeing players in their natural environment, displaying their actual, natural behavior, the further from "done" your design will seem to be. But this is what you want. You want a realistic read on how players will play your game in the wild, even though the closer you get to it, the more brutal the feedback becomes. It's horrid. Your game isn't fun. Players hate it. Maybe you should scrap the whole thing and start over.

For many designers, the solution is simply to drink from the fire hose. Put your head up in the stream and just take the full blast of feedback in the face. Actually, this works reasonably well. Big, serious issues tend to be very obvious, and the designer can usually figure out how to modify the system to correct the problems. Rinse, repeat, iterate. The more times you can iterate on this cycle of playtesting and modification, the better you game will be. However, there are some things that can be done to make this cycle take less time and be more effective.

Figure 5.1 is an example presented by Mick West in his article "Pushing Buttons" for Game Developer. "All I did was add a simple 'watcher' class which would record the value of a variable (such as a button up/down state or a physics state or flag) every frame and then display this as a scrolling state graph across the top of the screen, with a separate line for each variable that was being watched. To this state graph I added an event recorder which recorded events (jump, land, fall, super jump, late jump and crouch) and displayed them on the graph as a vertical line,

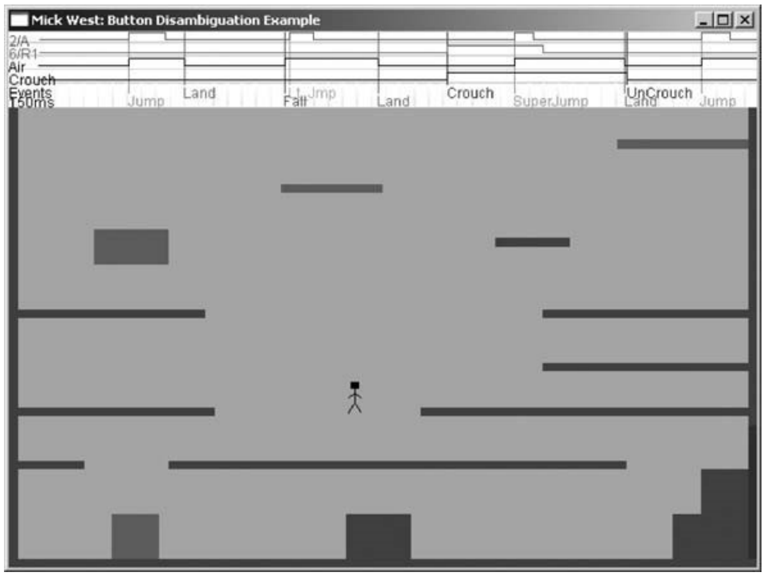

FIGURE 5.1 **Measuring inputs over time: an excellent "hard" metric.**

labeled with the event. Finally I made the graph able to be scrolled and zoomed in and out by the joypad when the game is paused. So, whenever some control problem occurs, it's very easy to pause the game and then scroll over and zoom into the area on the graph that caused the problems."

This is a sophisticated, data-driven measurement of player experience. West's millisecond-precise graphing of player input and system output provides a clear, quantitative explanation for a soft, wishy-washy, subjective player experience. This is great stuff. It enabled him to tune his system more quickly and to provide a clear, unambiguous insight into a very nebulous problem of the controls not feeling right for the player.

West's graphing is an example of a "hard metric." Hard metrics are quantifiable, finite measurements. The player pressed the button 57 ms after the computer thought they were off the cliff. The final score was blue team 10, red team 3. The player played the game for 27:03. Hard metrics provide specific, measurable data that can be compared across playtests. Assigning meaning to the data is part of the art of game design—should every game of Warcraft 3 take 20 minutes to complete? Or is it okay to have a game that last seven hours? Answering that question depends on the experience the designer intends to create.

Contrast hard metrics with "soft metrics"—things like fun, laughter and requests for more play. Are your players really having fun? What does fun mean in the context of your game? It could be deep strategic thought, with players sitting in silence, pondering the ramifications of their next move. Or it could be raucous laughter and intense interpersonal connections. Or it could just mean a feeling of relief and relaxation, a nice escape after a hard day at work. These things are not easy to measure. Though it is possible to quantify certain aspects of behaviors—such as Nicole Lazzaro's excellent Four Fun Keys, which are based on videotaped recording of people's faces and resulting emotional categorization—this is not the norm for game designers. Usually, soft metrics combine to form a sense, nebulous but always evolving, about what the experience of playing the game is for all players everywhere.

It's important to note that soft metrics are just as useful to game design as hard metrics. People tend to assume that because hard metrics are fact-based, scientific and objective, they are somehow better. But it's just as important to keep track of whether and how people are enjoying themselves while playing the game. Examining soft metrics is part of the game designer's intuition, which gets honed as he or she completes more and more designs. To have an intuitive grasp of what system dynamics will create enjoyable, meaningful experience is to be keenly attuned to soft metrics. In our measurement of the game feel of various games, we will employ both soft and hard metrics.

What's Important to Measure

In the Game Feel Model of Interactivity, it is the pieces on the computer's side of the system that can be changed by the game designer. Out of these, certain aspects are obvious candidates for metrics. The six most useful aspects in this respect—the most important to measure, in terms of framing principles for designing game feel and for comparing games—are as follows.

- Input—The physical construction of the device through which player intent is expressed to the system and how this changes game feel.
- Response—How the system processes, modulates and responds to player input in real time.
- Context—The effect of simulated space on game feel. How collision code and level design give meaning to real-time control.
- Polish—Effects that artificially enhance impression of a unique physical reality in the game.
- Metaphor—How the game's representation and treatment change player expectations about the behavior, movement and interactions of game objects.
- Rules—How arbitrary relationships between abstracted variables in the game change player perception of game objects, define challenges and modify sensations of control.

These are summarized in Figure 5.2. Some enable data-driven "hard" metrics, such as input, and others are on the "softer" side, such as metaphor. The rest of this chapter introduces these six elements as metrics in a general way. Chapters 6 through 11 describe the metrics associated with each element in detail.

Armed with this information, we can quantify game feel in a way that lets us design game feel from first principles instead of from *a priori* knowledge (i.e., by emulation). We will also have a detailed set of criteria for comparing games.

Input

The input device is the instrument of expression for the player into the game world. Therefore, the physical construction of the input device is important to the feel of control. The layout of inputs on the device, the tactile feel of the materials it's made from, its weight, and the strength of springs in joysticks and other actuators—all of these things affect the way it feels to hold, touch and use the input device. This changes game feel. It's like a musical instrument: while it's possible to play Fur Elise on a Playskool piano, there's a much greater potential inherent in a Steinway Grand Piano. When I create a prototype of a new control mechanic, it will almost always feel better to control using my wired Xbox 360 controller than using just buttons on the keyboard. At the highest level, this is because the Xbox controller is a well-designed consumer product made of sturdy materials and smooth, porous plastic.

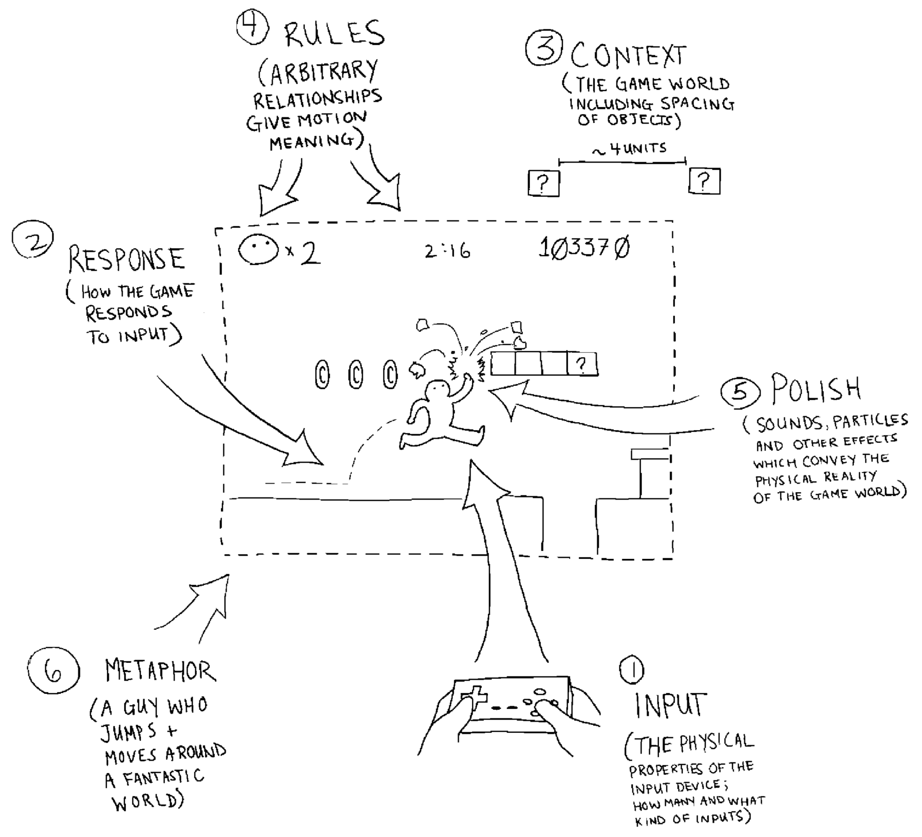
④ RULES
(ARBITRARY RELATIONSHIPS GIVE MOTION MEANING)
③ CONTEXT
(THE GAME WORLD INCLUDING SPACING OF OBJECTS)
~4 UNITS
?
?
② RESPONSE
(HOW THE GAME RESPONDS TO INPUT)
x 2
2:16
1Ø337Ø
?
⑤ POLISH
(SOUNDS, PARTICLES AND OTHER EFFECTS WHICH CONVEY THE PHYSICAL REALITY OF THE GAME WORLD)
⑥ METAPHOR
(A GUY WHO JUMPS + MOVES AROUND A FANTASTIC WORLD)
① INPUT
(THE PHYSICAL PROPERTIES OF THE INPUT DEVICE; HOW MANY AND WHAT KIND OF INPUTS)

FIGURE **5.2 The six most important elements of game feel to measure are input, response, context, rules, polish and metaphor.**

Playable Example

If you want to experience the difference and happen to have an Xbox 360 controller hooked up to your PC, try playing example CH05-1. You can control the game with either the Xbox controller or the keyboard (WASD.) Controlling the object with an Xbox controller feels better, all other factors being equal.

From the designer's side of things choices about which parts of the input device to use and how to use them affect the feel of control over virtual objects. The designer rarely gets to choose which input device the player will use—this is almost the same as making the choice about platform—but the designer can always choose which inputs on the device will be valid and useful for controlling things in their specific game. If the input device is a Playstation 3 controller, does the player use the thumbsticks, the buttons or both? These decisions define what sensations of control are possible in the game.

By choosing which inputs will be used, the designer also makes a choice about the sensitivity of their input space. Input devices have an inherent sensitivity. The feel of using an NES controller to control something in a game is different from a typical computer mouse, for example. The NES controller has more total inputs than a mouse—eight separate buttons—but is less sensitive overall than a mouse. Of these eight buttons, six are commonly used for real-time control, and each of these buttons is a very simple two-state affair. It's either on or off at any given time, nowhere in between.

The mouse has two standard buttons. But it also has a rolling ball or laser mechanism that recognizes movement in two directions. This two-axis movement is much more sensitive than a single button. The signals it sends to the computer are much more complex than the simple ON or OFF of a button.

From the designer's perspective, an input device translates the complex goals and intentions of the player into a simple language a computer can understand and interpret. This language is a stream of values that change over time. The movement of a thumbstick to the left, for example, can be interpreted by the computer as a change in a single "float" value—a number between −1.00 and 1.00. As the player moves the thumbstick, the numbers change. A button is much simpler in terms of physical activation and in terms of the signals it sends. Having selected a specific input device, the game designer then chooses which inputs on the device will be used, and how they will be used. In other words, of the possible input space inherent in the physical construction of the input device itself, the designer decides which parts will be valid and applicable to control in his or her specific game.

To measure the effect these choices have on game feel in completed games, we want to look at the properties of each individual input and at the input space as a whole. As a whole, we want to know which inputs on an input device are used for control, and to keep track of any physical constraints and limitations of the input device. For example, on an NES controller most of the inputs can be combined with one another, except for opposing directions on the directional pad. It's impossible to press both left and right on the directional pad at the same time. Ditto up and down. But it is possible to press the A-button and left at the same time. This means that a game controlled by a directional pad will feel very different than the same game controlled by four keyboard keys simply because the inputs combine in different ways (see Figure 5.3). On a keyboard, it *is* possible to press left and right at the same time.

For each individual input, it's useful to examine how many possible states the input has, the degrees of freedom and types of movement it permits, and how, if at all, it's bounded. For example, a simple button on my Xbox 360 controller has two states, on and off. It moves in one axis, up and down, and it is bounded in two places, at its maximum and minimum. A trigger button on the same controller still moves along only one axis but has hundreds of discrete states between its boundaries of fully released and fully pressed. The thumbsticks on the same controller have complete freedom of movement along two axes, X and Y, and are bounded by the circular plastic casing of the controller, giving them almost unlimited possible states.

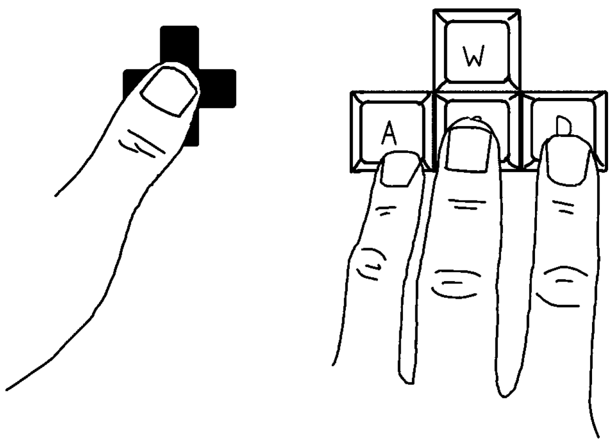

FIGURE **5.3 On a keyboard, it is possible—and likely—for a player to press left and right at the same time.**

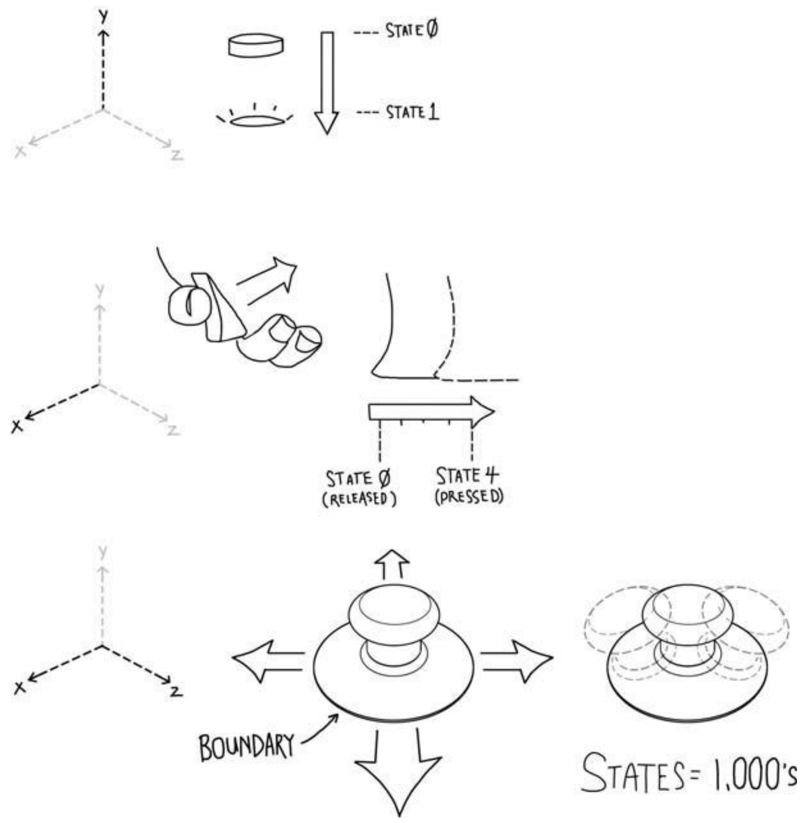

FIGURE **5.4 Differences in sensitivity between input devices.**

We can say then that a thumbstick is overall a more sensitive and expressive input than a trigger button, which is in turn more sensitive than a standard button.

Another metric, though a soft one, is how the input provides proprioceptive feedback (spring strength, layout of buttons relative to one another). Joysticks, thumbsticks, triggers and buttons can feel very different depending on the strength and quality of their spring mechanisms. This physical feel of interaction is very difficult to quantify, but has some effect on the feel of control over virtual objects.

Response

Knowing all about the input device is half of real-time control. All inputs eventually become signals, which are mapped to the modulation of some parameter in the game. This modulation can be thought of as the game's response to input.

The game receives signals from the input device. Signals modify some parameter in the game in some way, which is defined by the game designer. This is mapping: hooking specific input signals up to parameters in the game and defining how the parameters will be modulated over time. As Mick West says, "On the face of it, this appears a simple problem: you just map buttons to events, [but] getting player control to work is inevitably a fiddly and complex task."[2] This is where most of the feel of control in a game is created.

An input signal can modulate any parameter in a game and can do it over time in many different ways. A button press can move an object a distance in a direction. Press the button once and a cube moves five units to the right, for example. Or a game can continuously move that object a small amount each time the feedback loop is complete, as long as the button is held. Alternately, holding the button could add a force to a simulation, which indirectly causes the cube to speed up and start moving. All of these are different mappings of one button to the movement of one object, and with each, the feel is different. But a press of a single button can also be mapped to changes in global parameters. Pressing a button might reverse gravity or holding it might change the friction value for a car's tires, enabling a different kind of control.

To measure the response of a particular game, we want to look at how each signal coming in from the input device is mapped to a change in the game. What parameter does it modulate, and how does it change that parameter over time? Or, more generally, what parameters are changed by what inputs, and what are the relationships between the parameters? For example, in Id Software's Quake, rotation of the avatar in 3D space is very closely translated from the mouse on the desk surface. Changes in the two signals coming in—X and Y for the left/right and up/down movement of the mouse—rotate the avatar left, right, up and down (Figure 5.5). There is very little processing of input. The movement of the mouse adds a value to

[2]http://cowboyprogramming.com/2007/01/02/pushhing-buttons/

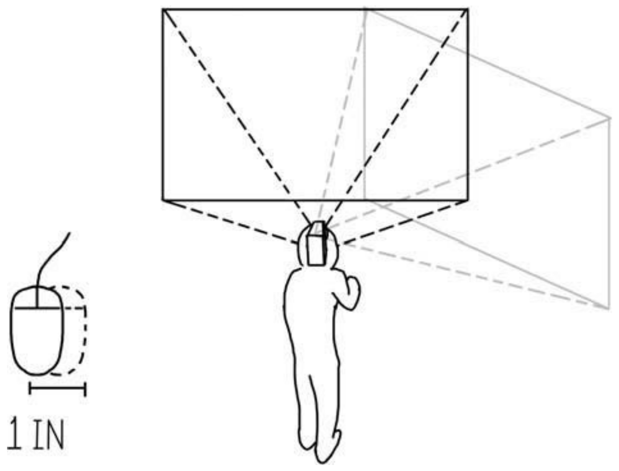

FIGURE **5.5 Moving the mouse in Quake rotates the avatar—input becomes response.**

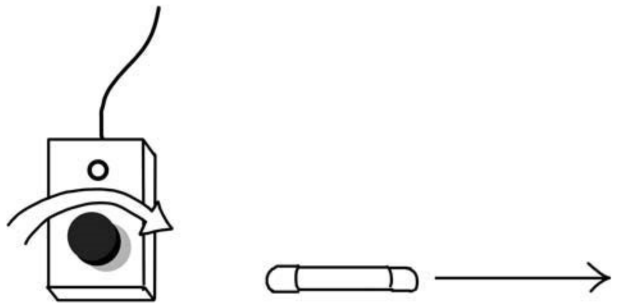

FIGURE **5.6 In Arkanoid, rotating the input knob maps to the left-right movement of the spaceship.**

the current rotation value of the avatar. Move the mouse an inch to the left and you turn 90 degrees to the left, a fixed steering ratio.

But in Quake, the steering ratio between horizontal mouse movement and horizontal in-game rotation is different from the one between vertical mouse movement and horizontal in-game rotation. For your up/down mouse movement, you get much less vertical rotation in the game. The relationship between these two values is just as important as the values themselves; in the context of Quake, the players will want to adjust their horizontal aim more quickly and over a greater distance than their vertical one.

Now compare the mapping of Quake with the classic Arkanoid, which maps rotation of a knob to the left and right movement of the in-game "paddle" spaceship (Figure 5.6). This is again mapped directly, though instead of mapping a linear movement to a rotation as Quake does, Arkanoid does the opposite, mapping the

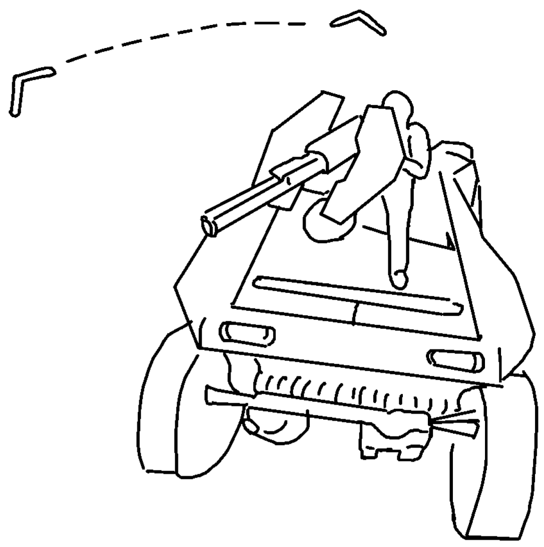

FIGURE **5.7 In Halo, the degree of change of the thumbstick moves the reticule at varying degrees of speed.**

rotation of the knob to a linear movement of the ship. One degree of rotation of the wheel input is translated into a certain distance of movement for the Arkanoid paddle.

Contrast this with the "warthog" driving mechanic in Bungie's Halo, a more indirect mapping of input to motion. Moving left on the thumbstick changes the position of the "reticule." This maps thumbstick displacement to a rate of movement instead of to a change in position. Moving the thumbstick to the left a small amount will move the reticule to the left at a slow rate. Pulling the thumbstick as far to the left as it will go moves the reticule quickly, at a constant, maximum rate.

The reticule's position represents a heading in the 3D game world, one which the warthog vehicle will then attempt to seek on—imperfectly. Depending on the distance between the reticule and the actual heading of the avatar, it will rotate more or less per frame to try to return to being in line with the reticule, but it may overshoot or undershoot. This obfuscation between intent and reaction is pleasurable rather than annoying because it defines both an interesting challenge and a pleasurable sensation of control.

An input signal can also be mapped to a modulation across time, such as the jumping mechanic in Super Mario Brothers. When jumping in Mario, holding down the button longer yields a higher jump. There is a maximum height and an expressive range in between. A tiny tap on the button is a tiny little jump. To get a full height jump, you must hold down the button longer.

After looking at what parameter each input modulates in the game and how it changes it over time, the final thing to examine is the relationships between the various parameters. No mechanic is an island. Just as important as how each individual mechanic is defined—how each specific input is mapped to a response—is the relationship between them. For example, the feel of Super Mario Brothers relies on the relationship between how fast Mario can move on the ground and how fast he can move in the air. In the air, he moves left and right, but more slowly than on the ground. You still have some control over Mario's trajectory in the air, but the feel is one of precise adjustment, giving you a better shot at landing on that small platform. The same kinds of relationships between parameters define the feel of any system of real-time control, from the speed-to-turning-radius relationship in a driving game to the size/speed tradeoffs present in most fighting games. As much as the basic mapping values themselves, it is the relationships between parameters that makes a game feel the way it does.

Context

Context includes simulated space and level design. Envision yourself playing Super Mario 64. Now imagine that instead of being in the middle of Bomb-Omb Battlefield, Mario is standing in a field of blank whiteness, with no objects around him. With nothing but a field of blankness, does it matter that Mario can do a long jump, a triple jump or wall kick?

If Mario has nothing to interact with, his acrobatic abilities are meaningless. Without a wall, there can be no wall kick. In this way, the placement of objects in the world is just another set of variables against which to balance movement speed, jump height and all the other parameters that define motion. In game feel terms, constraints define sensation. If objects are packed in, spaced tightly relative to the avatar's motion, the game will feel clumsy and oppressive, causing anxiety and

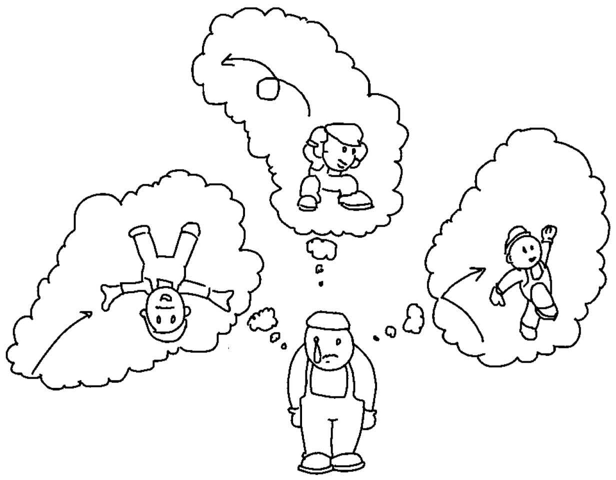

FIGURE **5.8 Mario is sad because he has no context for all his fancy moves.**

frustration. As objects get spaced further apart, the mapping of input to response becomes increasingly unimportant. It doesn't matter how fast a car moves relative to its turning speed when it's driving across a featureless landscape. Of course, the spacing of objects is irrelevant until two objects can interact. It could look as though you're driving in a densely packed forest, but if the car goes through the trees, it makes no difference how many trees there are or how they're laid out. In this sense, collision code is the other part of context.

Context, then, is the unique physical reality of the game world—the simulated space—including the way that objects interact and the layout of space. Like the abilities and actions of the avatar, it is designed. The game designer creates a game space that has its own unique physics, extents and constraints. The designer simultaneously creates the content that fills that world and defines its spatial relationships.

Almost every game has a contextual aspect of some kind, be it tracks in Gran Turismo or tracks in Guitar Hero. Tracks, puzzles, stages, levels, worlds: most games have some kind of designed context against the mechanics' functions. In most cases, this is called level design. The objective is to find the most interesting pieces of the mechanic and emphasize them by trying to provide the most interesting interactions possible with the mechanics.

The importance of this context will vary depending on the type of game, but almost every game includes some kind of level design. My favorite example is the difference between Slim Tetris and regular Tetris, as shown in Figure 5.9. Level design isn't as important in Tetris as it is in, say, a Tony Hawk game, but Alexei Pajitnov still

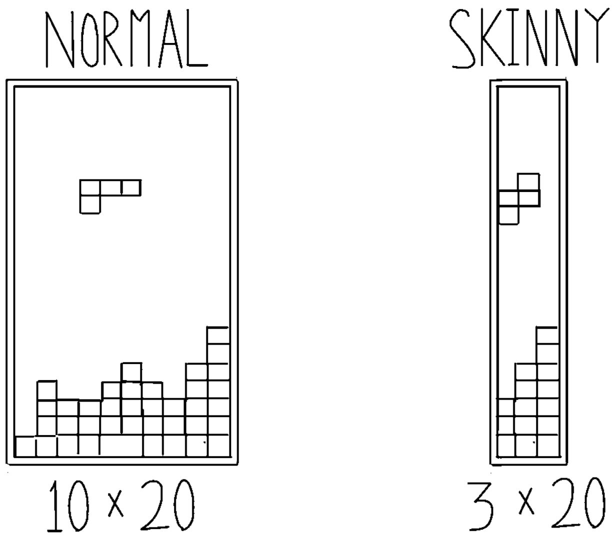

FIGURE **5.9 Normal versus Skinny Tetris: if the level design was different, it wouldn't be the same game.**

had to decide that the Tetris field would be 10 blocks wide by 24 blocks tall. If it were three blocks wide, Tetris would be a very different game. Change the context, change the game.

The default playfield of Tetris provided the right balance between constraint and openness. It was an artistic choice and reinforces the important aspect of context: spatial constraints define challenge. The sharpness of turns and spacing of obstacles in Mario Kart DS defines the challenge of a particular course. The more difficult courses have sharper turns that happen more frequently and include more obstacles, jumps and other challenge-making context.

The effect of context on game feel can only be expressed as soft metrics. There is a change in the sensation of control when the overall landscape of the game world is huge and open versus when it's constrained and claustrophobic, but the exact change is not quantifiable. It's a general impression of space. The feel of control also changes when objects are spaced closer together or further apart, and when objects are sharp, round, organic or blocky. If you bump into things all the time, the game feels different. At the lowest level, collision code again redefines feel as it makes interactions feel a certain way. The interaction of objects can be smooth, as they slide off one another or feel tacky, if they stick when they come into contact. If an object explodes instantly when it runs into something, this again changes the feel of the game. Steering around objects becomes much more important, and the thing you're controlling seems fragile.

The best way to measure these effects is to examine the feel of control in different contexts within the same game. In Asteroids, there are times when there are as few as one asteroid on the screen. When that happens, the feel of control is different from when the screen is covered in tiny, high-speed asteroid fragments. In these extremes, we see the way that the feel of Asteroids is changed when the playfield—the spatial topology—goes from empty to full.

Polish

Polish is any effect that enhances the interactions between objects in the game world, giving clues about the physical properties of objects. If all polish effects were removed, the functionality of the game would be the same, but the player's perception of the physical properties of the objects in the game would change. Perception is active, and polish effects further define the interactions that occur because of a game's collision code.

Measuring polish is another soft metric. How does the feel of De Blob (the student game mentioned in Chapter 1) change when the squash-shader is applied? This is not quantifiable as a hard metric. What we can measure is the resulting impression of physicality. Specifically, what the polish effects seem to tell us about the properties of the objects we're observing. The blob squashing and stretching is what makes it seem a blob. As Joost says, without the squash-shader, the game feels like playing with a ball made of stone.

This principle applies to all polish effects. All effects, even if they're applied only with the nebulous goal of making interaction more appealing, send signals about the physical properties of the objects involved in the interaction. When it's passive—when you see two objects in the distance collide—the impression is the same as in film or animation. When it's active, the impression is much more powerful, like experiencing something with your own senses.

A violent spray of particles, the screen shaking or a loud noise lend the impression of weight, heft and solidness to objects. Any interaction that can happen in real life, that has happened in film, or that can be imagined can be conveyed through polish effects. The thing that's interesting to measure is how the polish effects impact a player's perception of the objects in the game world.

Playable Example

See example CH05-2 for some different polish effects applied to the same system.

Consider the game Burnout: Revenge, a game with an almost preposterous amount of polish. A car in Burnout is a solid object traveling at high speed. When it crashes, the force of the impact is palpable, and the results are catastrophic. The car is deformed in a hail of glass and sparks. Its mass and high velocity carry it high into the air, spinning and burning, eventually to crash back down with all the weight of its two-ton frame. The crunching, shattering and scraping noises are gut-wrenching. A car in Burnout is a solid object traveling at high speed which has now been completely obliterated (Figure 5.10).

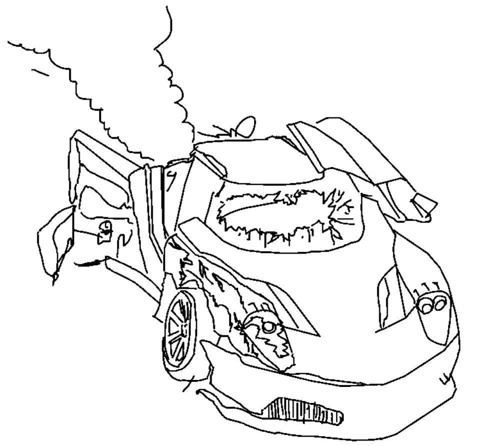

FIGURE **5.10 The cars in Burnout can be damaged to an amazing degree. This is some serious polish.**

But how do you know this? What tells your senses that this is the case? How can you know the physical nature of this digital object? What clues are you using to derive this understanding? Well, let's break it down a bit. First, we have the visuals: glass is spraying from the windows, pieces of metal and car parts are flying off in every direction, and dust and smoke are spewing from the engine. These are probably simple particle effects—two-dimensional images displayed at a particular point in the 3D scene but which are programmed to always be facing toward the camera. Often, these are a series of frames which play back linearly with some randomization, causing little pieces of glass to spin and puffs of smoke to appear to billow and froth. The sparks flying from where the car is contacting other pieces of metal or scraping against the divider or pavement are probably generated the same way, transitioning in color from white to yellow to red over time as they spray out. The tires leaving skid marks on the pavement are probably alpha textures, being laid down with created-on-the-fly geometry that has an alpha-blended tire tread texture mapped to it. Maybe it has two or three different layers and randomizes the texture so it's hard to see the textures repeat.

And what about the sounds, the screeching, the skidding and the shattering glass? The sounds of rending metal are all created, blended and triggered in real time. They even have locations in 3D space, using positional audio to further emphasize the link between sound and visuals. And then there's the controller rumble, adding a little bit of tactile sensation to the mix. It may not be especially logical, but it helps the impact seem more, ah, impactful.

So where did all these clues come from? Did a game designer simply press a button that says "insert car" and knock off down to the pub for a pint or two? Sadly not. Each tiny piece of interaction, each particle effect and sound, each deformation and broken piece of car sent flying is a hand-crafted response meant to do one thing: convey the physical nature of this interaction to you, the player. It must be a combination of various sights and sounds, too, because perception is a multisensory phenomenon. When you perceive things, you see, hear, touch and feel all at once. Perceiving something involves your entire body, even when it's an extension of your sense into a virtual body. More than that, your perception of something includes the meaning you assign to it based on past experiences, ideas, feelings and generalizations. If it looks like a car, you expect it to behave the way things that fit your idea of car would behave. This might be your experience of a real car crash, years of watching car crashes in movies, or both. The point is that the clues must be designed by a designer, created by an artist and programmed by a programmer. Often, all three will touch a complex effect. Often this is written off as "just polish" but as applied to game feel its impact in terms of conveying a convincing, self-consistent game world can't be ignored. Polish is important.

Metaphor

Metaphor is where a player's past experiences, ideas, feelings and generalizations come into play. Not only from playing games, but from their total life experience.

What does the thing you're controlling look like, and how do you expect it to behave based on your experience with similar things? If it looks like you're controlling a real car, the expectation is that it will handle like a car, sound like a car and crash like a car. But the expectations come from the idea of what a car is, not from objective reality. Players bring with them all their life experience—riding in cars, driving cars and so on—but they also bring their experiences of cars from films and animations. Convincingly behaving like a car, then, might mean exploding after being shot with one bullet or it might mean squashing and stretching as in a cartoon, taking no visible damage. Oftentimes, people will play a game—horse-riding gameplay is my favorite example—and they'll say, "This doesn't feel like a horse." And you'll ask them, "Well, have you ever ridden a horse before?" And they'll say, "No, but this doesn't feel like a horse." What this illustrates is that players carry with them preconceived notions about the way certain things move and, by extension, how it should feel to control them.

The expectations about how interactions should play out are also influenced by treatment—how the art is executed. A cartooned, iconic car has much more leeway when it comes to how it can behave and interact than a photorealistic one.

Try this as a thought experiment: instead of the car in Burnout, substitute a giant, balding fat guy running as fast as he possibly can, spraying sweat like a sprinkler in August. Without altering the structure of the game, the tuning of the game or the function of the game, the feel of the game has changed. All you've done is swap out a 3D model of a car for a 3D model of a giant fat guy running and you've got *Run Fatty Run* instead of *Gran Turismo*. This will change the feel of the game because you have preconceived notions about the way a car should handle.

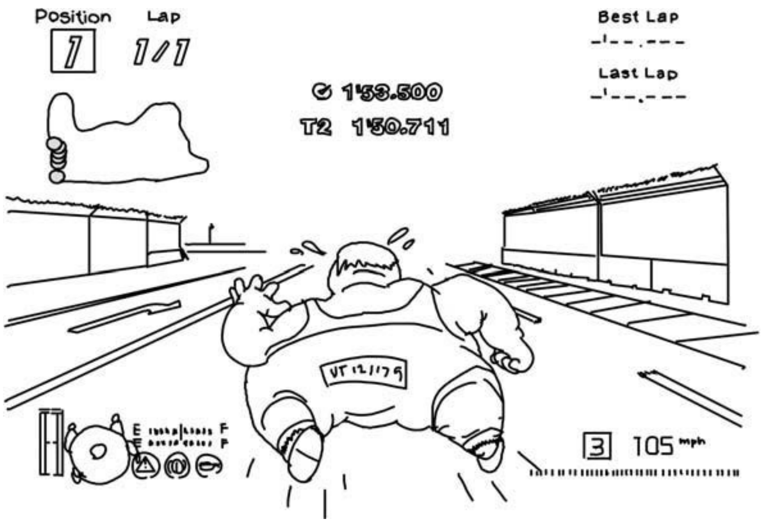

FIGURE **5.11 Run, Fatty, Ruuuuuuun! This avatar sets up different expectations than a car does, which changes the feel of the game even if the underlying functionality is identical.**

You know how a car should feel and move and turn based on your experience driving a car and looking at cars. When thinking about a game feel system, it's important to understand the palette of preconceptions you're working with. The best designers use metaphor and treatment to set up expectations in the player that can then be exceeded by the game's interactions.

Rules

Returning to Super Mario 64, have you ever asked yourself "Why am I collecting these coins?" If it didn't refill Mario's health or if 100 coins didn't get you a star, would you bother? Would it be worth it? For that matter, why is collecting a star important? What "value" do these things have? Outside the system of the game, none whatsoever. They are abstract variables whose arbitrary relationships give them value within the cohesive whole of the game system. In other words, the meaning of a coin, a star or any other such part of a game is given only by its relationship to other parts of the game. It's manufactured from nothing. From thin air. Poof. It's a system that gives itself meaning. Isn't that weird? It works, though. You want those coins, and you want that star, and you're willing to undergo a lot of frustration, tedium and learning to get them. The intrinsic pleasure of learning and doing may be the fundamental appeal, but it's the carrot of the stars that gets you moving.

Traditionally, this is the role rules play in both game design and in game feel. They provide motivation and a structured way to learn, defining for the player the motions that are worth learning. Indeed, it is the seemingly arbitrary relationships between variables that give the motion meaning. A gradual ramp of increasingly difficult challenges matches the player's growing skill, keeping them in the flow state (Figure 5.12), introduced in Chapter 1.

In the context of game feel, rules as we've defined them provide motivation, challenge and meaning for motion. Context provides the immediate, spatial meaning while rules provide the long-term, sustainable meaning that games are built out of.

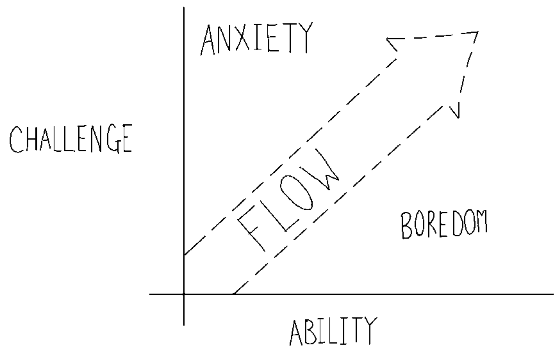

FIGURE **5.12 Csikzentmihayli's Flow.**

As in our earlier example, rules provide some, if not all, of the intent in a game system, as shown in Figure 5.13.

Race from point A to point B, scale this tall mountain, rescue five wayward puppies, escape the compound. These kinds of higher-order goals define game feel at the level of sustainability. They operate at multiple levels, with multiple goals and types of goals active at any given time. This promotes a high level of engagement and gives the player many choices about which activity to pursue at a given time. The low-level sensation of control and physicality, which we've defined as game feel, is a great foundation for quality game experiences, but it's the higher-order rules that provide the girders and scaffolding to build it out.

To measure the effect of rules have on game feel, we can look at rules in three different ways. Again, these are soft metrics, as they are not measuring specific quantities. What we're interested in is how seemingly arbitrary relationships between variables can change the meaning players assign to objects in the game world, changing the feel of control and interaction as they perceive that world.

At the highest level, goals focus the player on a particular subset of motions. These high-level goals provide a trickle down effect, giving objects meaning at various levels. High-level rules can also be things like health and damage systems, which again trickle down to give meaning to moment-to-moment interactions.

Separate from but hooked into the high-level rules and goals, mid-level rules can give meaning to objects in the game world, changing the feel of moving through it.

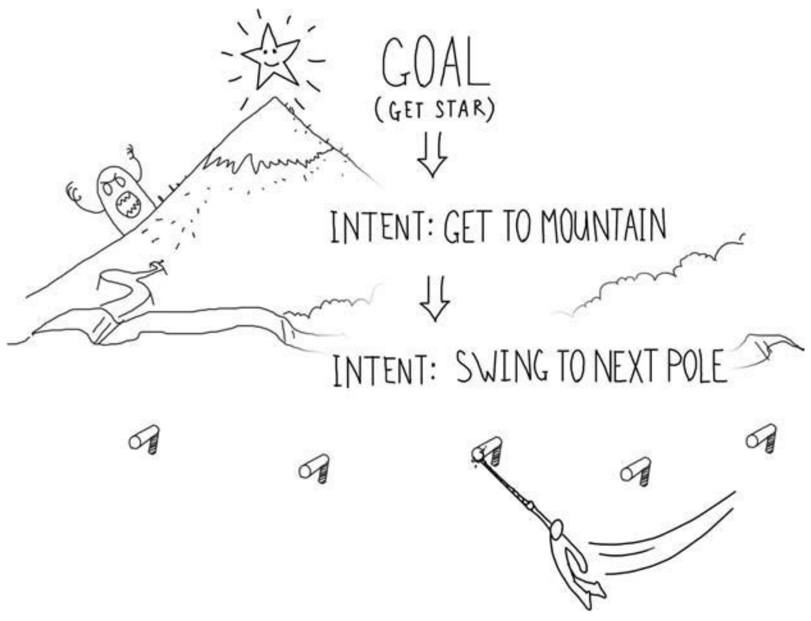

FIGURE 5.13 **High-level goals trickle down.**

The flag in a capture the flag multi-player game is one example; for the player currently holding the flag, the game feels different.

At the lowest level, rules can further define the physical properties of objects. How much damage it takes an avatar to destroy an enemy changes the player's perception of how "tough" that enemy is. An enemy that takes one hit to destroy will feel fragile, while a boss monster that takes 20 hits feels much more solid.

Summary

The six pieces of the game feel system that are malleable for the game designer are:

- Input—The physical construction of the device through which player intent is expressed to the system and how this changes game feel.
- Response—How the system processes, modulates and responds to player input in real time.
- Context—The effect of simulated space on game feel. How collision code and level design give meaning to real-time control.
- Polish—Effects that artificially enhance impression of a unique physical reality in the game.
- Metaphor—How the game's representation and treatment change player expectations about the behavior, movement and interactions of game objects.
- Rules—How arbitrary relationships between abstracted variables in the game change player perception of game objects, define challenges and modify sensations of control.

For each of the six pieces of the game feel system, I've pointed out a few different things that are instructive to measure when examining a particular mechanic or a particular game feel system.

Each of these is discussed in more detail in Chapters 6 through 11. We'll be looking at what can be measured and what's useful to measure. We will be pursuing both soft and hard metrics. For each measurement, we'll go through why this is useful to know about a particular game and how it helps us compare the feel of two games in a meaningful way.

The point of measurement is to derive general principles about game feel which can be applied to future designs and to let us meaningfully compare the feel of two games. Instead of taking shots in the dark, emulating existing mechanics or trying to shoehorn someone else's tuning into your system, you want to be able to understand the tools at your disposal. If you want your game to feel like Sonic, Megaman or Burnout: Revenge, you'll be able to do it with a deeper understanding. You might not have the exact recipe—it's probably secret—but at least you won't be staring at a finished cake wondering what kind of sugar was used.